NASA Contractor Report 159229

INTERMITTENT/TRANSIENT FAULTS IN COMPUTER SYSTEMS--EXECUTIVE SUMMARY

Gerald M. Masson

THE JOHNS HOPKINS UNIVERSITY
Baltimore, Maryland 21218

NASA Grant NSG-1442
April 1980

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

Executive Summary

Intermittent/Transient Faults in Computer Systems

NASA Grant NSG 1442

Gerald M. Masson
Electrical Engineering Department
The Johns Hopkins University
Baltimore, Maryland 21218

Contents

Preface

To determine the capability of a fault tolerant computing system to diagnose failure situations such that transparent reconfiguration can take place in the sense that the identified failed elements of the system are removed and the remaining operable resources are redeployed is a necessary, fundamental component of the validation process [1]. To make this contribution to the validation process requires the following:

(1) An abstract model of the system which can describe the essence of the testing strategies employed by fault tolerant computing systems such as SIFT or FTMP.

(2) An abstract fault model which can describe the broad range of failure situations to which a system can be subjected.

(3) A theory which utilizes the system model and fault model so as to characterize the conditions which must be satisfied by a fault tolerant computing system such that various levels of diagnosis quality are achieved.

(4) A methodology which is demonstrated to be feasible and which describes fault to error transformations such that the above models and theory can be applied to the diagnosability analysis of fault tolerant computing systems.

In this Executive Summary, we consider each of these four points to show that the program being supported by NASA at Johns Hopkins represents a total, comprehensive approach to the intermittent/transient fault issue. We feel our work is a complementary theoretical and experimental effort which will eventually result in a fundamental validation tool for use in a clinical environment such as NASA's AirLab.

Executive Summary

Intermittent/Transient Faults in Computer Systems

1. A Contribution to the Validation Process

Fault tolerant computer systems are currently being designed to realize specific reliability requirements. The proof that any specific design actually satisfies these reliability requirements is referred to as validation. Currently, a number of approaches are being explored which have as a goal the development of tools to contribute to the validation process [1]. The following is an overview of the approach being taken at the Electrical Engineering Department of The Johns Hopkins University. Our overall goal in this effort is to develop an interrelated theory and experimental methodology which can be used in a laboratory situation to measure the capability of a fault tolerant computing system (such as SIFT or FTMP) to diagnose intermittent/transient faults (I/T faults). Such diagnostic capability is fundamental to the transparent reconfiguration process of fault tolerant computing systems by which faulty subsystems (called units) are identified and the remaining operable resources are redeployed. The experimental and theoretical work we are doing addresses the problem of developing an abstract model (with fidelity to SIFT and FTMP structures) for which necessary and sufficient conditions can be established which must be satisfied to achieve various diagnosing capabilities. The application of this abstract model to a specific system requires the availability of data which must, therefore, be generated in a controlled laboratory situation for the specific system under consideration. Hence, a verified, well-defined methodology for generating this data is being established. To the extent that such diagnosing capability is important to reliability in fault tolerant computing systems, this theory and supporting methodology will serve as a foundation for validation efforts.

2. A Systems Level/Error Level Approach

It must first be understood that all current designs of fault tolerant computing systems are large configurations of computationally and logically interconnected subsystems, called units. These units can be hardware, firmware, or software subsystems (or combinations); and it is their complex interactions and responses to data which constitute system operation. When describing the operation of a fault tolerant computer, it is, therefore, this system level view of interacting units that must be modeled; for if, say, a logic circuit level or instruction word level view were modeled, then when an attempt was made to develop an overall systems model, the details of each composing subsystem model would compound to produce such overwhelming complexity that the development of a comprehensive systems model would be impossible.

Similarly, regarding the capability of a fault tolerant computer to diagnose classes of faults, while it is important to study particular types of circuits or instruction executions so as to characterize their faulty operation or effects in the presence of I/T faults, these low level approaches are too dependent upon the particular circuit or instruction word involved to contribute of themselves to the goal of systems level assessments for the validations of fault tolerant computing systems. In other words, such low level results are an end in themselves; they are a useful input to validation only if a developed process or technique which has been generally established can utilize them. Accordingly, since it is our goal to develop a theory and experimental methodology which will use such results in contributing to validation, we must look at the faulty operation of units at a level which is in agreement with our systems level view of the overall design. That is, we must consider faults at the error level. As this complements our systems level model of fault tolerant computers, it gives us a framework for the development of our theory and the establishment of our supporting experimental methodology which is not tied to the specifics or peculiarities of any circuits or instruction sets.

3. A New Viewpoint of I/T Faults

All of our results (past and current) have been achieved because we have taken a fundamentally new view of I/T faults. Heretofore, researchers have viewed I/T faults as special cases of permanent faults. That is, an I/T fault in a computer system had previously always been viewed as being equivalent to a permanent fault except that it somehow turned on and off. In effect, the I/T fault case was always treated as a more difficult to handle, elusive version of a permanent fault. Because of this, all theory, analysis tools, and fault tolerant computer designs that were developed for permanent fault situations were useless relative to I/T fault cases since they depended on the constant, repeatable exhibition of an error mode. When forced to consider I/T faults with these permanent fault based theory, tools, or designs, there was no recourse but to "simply hope to be lucky enough to catch the I/T faults acting like permanent faults".

We have "turned the table", so to speak, on this viewpoint. We, instead, view permanent faults as special cases of I/T faults. That is, we consider a permanent fault to be an I/T fault with a very long duration. The ramifications of this simple reversal in viewpoint are major because anything (that is, theory, analysis tools, fault tolerant computing designs,...) that has been accomplished or developed for the permanent fault case must now be a special case of what is necessary for the I/T fault case. This has permitted us to study permanent fault work as an indicator of what a degenerate version of the results we were seeking for I/T faults looked like. This was immensely useful because it meant we were not starting from absolute zero. Indeed, we had a good picture of a version of the answers we sought. Moreover, a potentially even more important ramification is now being investigated. It seems that it might be feasible to predict or measure the I/T fault diagnosis capabilities of a system based on the permanent fault diagnosis capabilities. This would mean that systems that have been studied or experimented with from a permanent fault point of view might not have to go through the process again for I/T faults; instead, the theory we are developing might be capable of extrapolating the existing permanent fault results to the I/T fault case.

4. A Formal Model for Diagnosability Analysis

Given our new viewpoint of I/T faults and that a systems level perspective is to be taken, a formal model with which to analyze the capability of a fault tolerant computing system to diagnose fault situations must be specified. We have chosen to use as the basis for our work a systems level model which is well-known and has received considerable attention in the fault tolerant computing research area over the past ten years. Moreover, the model is totally appropriate for distributed fault tolerant computing systems such as SIFT and FTMP. However, the previous research efforts (that is, the reported work that preceded our involvement in our NASA Grant) considered only the special case of permanent faults. One of our major contributions up to this point on this NASA grant has been our extension to the general case of I/T faults of many of the significant research results based on this model that have been achieved by previous researchers for the special case of only permanent faults. Some of our extensions have already been reported in the literature [2]; indeed, our previous extensions are now recognized as benchmark contributions to the theory of systems level diagnosability analysis.* Moreover, our current results are even more significant as we have now broadened the scope of the theory to include combinations of permanent and I/T failures in the systems. We refer to these as hybrid fault cases. This new hybrid fault class allows us to analyze fault tolerant systems where some of the units composing the system are known to not be susceptible to faults which cause I/T error modes, and it also allows us to consider the response of fault tolerant systems to unanticipated fault environments where combinations of the units assume permanent and I/T error modes.

In order to give an overview of some of our new results and our motivation for our current and future efforts, it will be necessary to briefly describe a (rather restrictive) version of our model. Given a fault tolerant computing system that is to be analyzed for its capability of diagnosing I/T faults (and, because of our view of I/T faults it will also be analyzed for the special case

---

*For example, two recent survey articles on systems level diagnosis and I/T faults have entire sections devoted to the results we reported in [2]. These two articles are: "Systems Level Fault Diagnosis" by A. Friedman and L. Simoncini, to appear in IEEE Computer Magazine, and "Intermittent Fault Analysis in Digital Systems" by S. Su, Proceedings of the 1979 National Computer Conference.

of all permanent faults), the system is partitioned into logically and computionally disjoint units (often referred to as least reducible units). This set of LRU's, or units, can be described, in general, by the set $U = (u_1, u_2, \ldots, u_n)$. Each unit, say, $u_i \in U$ is capable of testing other units in the system; however, a unit is assumed to not test itself. To perform fault diagnosis, each unit $u_i \in U$ is assigned a particular subset of the remaining units in the system to test. This assignment of testing tasks to units is referred to as the connection assignment of the system. The connection assignment is described by a set T of tests, where a test $t_{ij} \in T$ if and only if unit u_i tests unit u_j. The connection assignment can also be represented by a directed graph model, where there is a node representing each unit $u_i \in U$, and where there is a directed edge from a node u_i to a node u_j if and only if $t_{ij} \in T$. A test t_{ij} is said to have been applied when unit u_i performs a test on unit u_j, and we refer to a test set application as the application of each test $t_{ij} \in T$. Figure 1 is an example of our graph model. It consists of 7 units and each unit tests 3 other units.

In our model, the application of a test $t_{ij} \in T$ implies an evaluation of unit u_j by unit u_i to determine whether u_j is faulty or fault-free. The details of the component failures which can inflict the units are only of concern to us at the error level to the extent that each such failure can result in the inflicted unit behaving as either a permanently faulty unit or as an I/T faulty unit. A permanently faulty unit can generically be considered to be a unit which is in some fixed error mode of operation such that its functional behavior is different from the unit's error-free mode of operation. An I/T faulty unit can generically be considered to be a unit which can be at any given time in any of several possible error modes, or, indeed, even the error-free mode. A test, t_{ij}, can be viewed as a sequence of stimuli which is to be applied to u_j by u_i so as to elicit responses from u_j that are evaluated by u_i and which are sufficient for the detection of the existence of all the possible error modes of operation of importance in u_j.

If we assume for the moment that u_i is fault-free, it should be emphasized that in the permanently faulty case, since u_j is in a fixed error mode, the appropriate stimulus within the test t_{ij} will always be eventually applied so that the presence of the error mode will be detected. However, in the I/T faulty case, since the error modes which u_j exhibits can change or, indeed, since u_j can even be, at times, in the error-free mode, it is possible that as the stimuli within t_{ij} are applied, u_j might change its modes so that, at any time, the particular stimulus applied is not capable of detecting the error mode

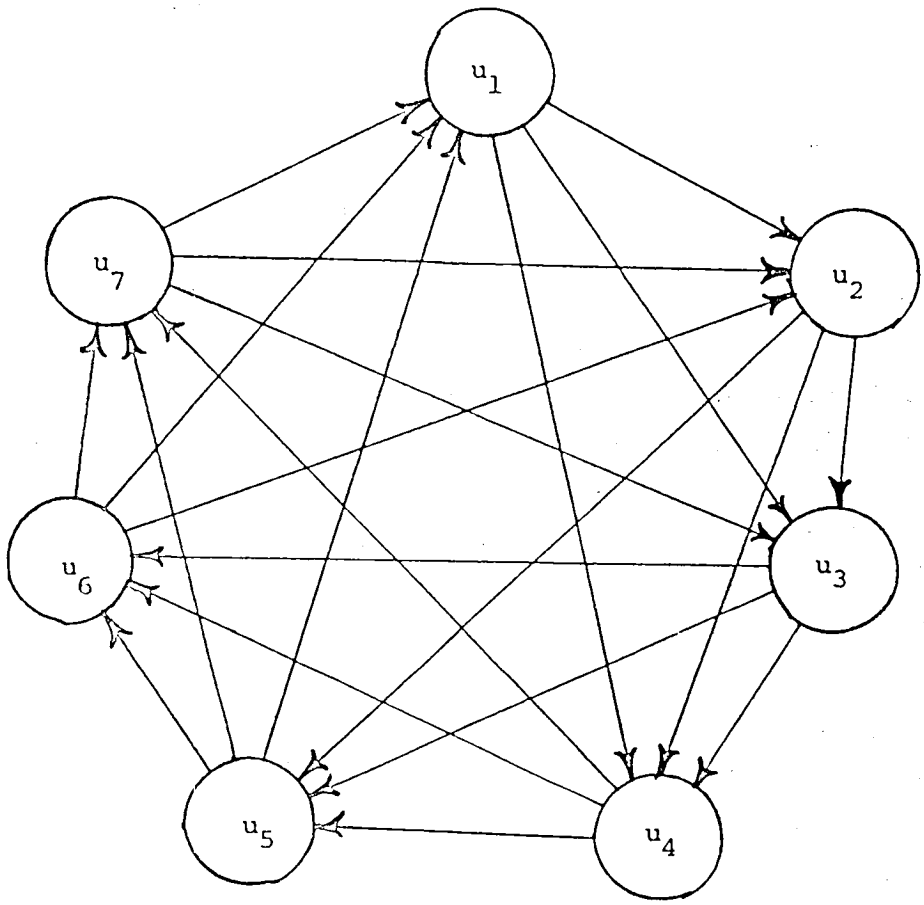


Figure 1: A PMC model of a system of 7 units which employs the $D_{1,3}$ design.

being exhibited by u_j at that time. If, in fact, this were the case, then the fault-free u_i would evaluate the I/T faulty u_j to be fault-free on the basis of this application of t_{ij}. This is the essence of the additional difficulty introduced by the existence of I/T faults beyond that caused by permanent faults.

As a means of expressing the results of a test, t_{ij}, in our model, we can associate an outcome a_{ij} to t_{ij}. A complete set of test outcomes, that is an outcome a_{ij} for each $t_{ij} \in T$, is called a syndrome. The different collections of outcomes that can be formed in the syndromes are indications of the data that can be used by the fault tolerant computing system to diagnose faults.

It should also be clear that the details of the connection assignment reflect directly on the amount of information that can be obtained from a test set application for the identification of faulty units. Accordingly, if a diagnosing capability of a system is desired for which a class of fault situations is specified and the manner in which sets of faulty units which fall into this class are to be identified on the basis of syndromes that can result from a test application is described, then conditions (necessary and sufficient) can be determined for the connection assignment which must be satisfied to guarantee the desired diagnosing capability. Given a system design, we can then analyze the connection assignment of the system to see whether a desired diagnosing capability is achieved or to measure what diagnosing capability is achieved.

To illustrate by using this version of our model the excessive difficulties created by I/T faults in fault tolerant computing systems, suppose that we are using the simplest outcome set in our model where for a test t_{ij}, a value of 1(0) is assigned to a_{ij} if u_i evaluates u_j as faulty (fault-free). We can now more specifically describe the scope of the test result possibilities when we have permanently and intermittently faulty units in our models. This is done by means of the Outcome Matrix shown in Figure 2. In general, tests can be performed by units which are fault-free, permanently faulty, or intermittently faulty. The Outcome Matrix then specifies the outcome of a test t_{ij} for all possible combinations of faulty and fault-free conditions for u_i and u_j. A justification for each of the entries in the Outcome Matrix can be given briefly as follows. If u_i is permanently faulty, then when t_{ij} is applied, it cannot be apriori specified whether u_i will evaluate u_j correctly regardless of whether u_j is permanently faulty, fault-free, or I/T faulty. In each of these cases it then follows that $a_{ij} = 0$ or 1. It is

u_i \ u_j	Permanently faulty	Fault-Free	I/T faulty
Permanently faulty	0 or 1	0 or 1	0 or 1
Fault-Free	1	0	0 or 1
I/T faulty	0 or 1	0 or 1	0 or 1

a_{ij}

Outcome Matrix

Figure 2

important to stress that when u_i is permanently faulty and u_j is either permanently faulty or fault-free, then although $a_{ij} = 0$ or 1 and cannot be apriori specified before the initial application, repeated applications of t_{ij} will always yield the same outcome as a result of the fixed error mode(s) involved; however, when u_j is I/T faulty, repeated applications of t_{ij} can yield different outcomes as a result of the different modes which u_j can exhibit. Next, if u_i is fault-free, then the entries for the cases where u_j is permanently faulty or fault-free are clear; and, again, because the error modes exhibited by u_j can change when it is I/T faulty, we have that in this case $a_{ij} = 0$ or 1, and the outcome can vary for different applications of t_{ij}. Finally, if u_i is intermittently faulty, $a_{ij} = 0$ or 1 for all possibilities for u_j; and, for all cases in general, the outcome can vary for different applications of t_{ij}. Now, refer to Figure 1. Suppose that units u_1, u_2, and u_3 were permanently faulty. To diagnose this all permanent fault situation, we would have (based on the Outcome Matrix) one of 512 possible syndromes to work with after a test set application. It can be shown that each of these syndromes would be sufficient for the diagnosis of this fault situation [2]. But now suppose instead that u_1 and u_2 were I/T faulty and u_3 were permanently faulty. There would then be 16,384 syndromes that could result after a test set application, and most (approximately 95%) would be useless for diagnostic purposes. In other words, this hybrid fault situation causes the production of many more syndromes than the permanent fault situation, and, moreover, most are of no help in diagnosis.

Many examples much worse than this could be presented here, but our point is that with our model we can illustrate such results and can furthermore show how the testing assignments in the system can be augmented to increase the I/T diagnosing capability. It should also be stressed that the above is based on a rather simple version of our model. We are currently developing a much more general model. This new model will to a still finer degree allow us to characterize the testing strategies of fault tolerant computing systems so that diagnosability analysis can take place. This is discussed somewhat in the next section.

5. Generalization of the Model to Broaden Applicability

Our model (in the form described in the previous section and its more general form) and the theory we have developed and are now developing around it make possible the diagnosability analysis of fault tolerant computing systems. However, there are restrictions in its applications. Note that the model thus described lumps all faults into either a permanent or I/T fault class. No distinction is made between different types of permanent or I/T faults. While this lumped fault class approach yields important results, because of its simplification of the actual fault situation, it does have restrictions in its applications since in practice there are many different classes of faults acting on a unit to make it function incorrectly. For example, because some of the faults may be easier to detect than others, by placing all of these faults into one category (I/T faults), potentially useful information regarding diagnosis has been lost. By expanding the I/T fault category into more specified and varied fault classes, additional information is made available which we should be able to use to improve the overall diagnosability analysis of the system. For example, tradeoffs could be considered regarding the time consumed in the testing procedures and the information obtained from each test, or (for the above form of the model) the number of links required by the network to achieve various fault diagnosis capabilities can be determined.

In considering possible generalizations of our fault model to increase the applicability of the diagnosability theory, suppose we specified a set of m fault classes: $F=\{f_0,f_1,\ldots,f_{m-1}\}$. These classes could be assumed to include both permanent and I/T faults, and f_0 may be considered to be the fault-free case. A straightforward expansion of the (3x3) Outcome Matrix would yield an (m X m) Outcome Matrix where all faults were listed in the rows and columns. But little would actually be gained by this extension since most of the entries would still be "0 or 1" . However, instead of restricting the value of a_{ij} to only 0 or 1 , suppose we let a_{ij} be a variable between 0 and 1 which corresponds to the probability that u_i when inflicted by a specified fault in F would assess u_j when inflicted by a specified fault in F to be faulty. Clearly, such additional information in the Outcome Matrix could be exploited in diagnosability analysis. Unfortunately, I/T fault situations are not simple enough to permit this straightforward generalization. In particular, a unit can react differently to the same fault depending upon the function or task it is

performing when the fault occurs. This was masked in the lumped version of the Outcome Matrix of Figure 2 as entries were made "0 or 1" if the outcome was not fixed. To permit individual fault classes to be considered, the task being performed must be taken into account. For our systems level view, the tasks are described as programs. All programs which the units will execute can be considered as the set of programs $P=\{p_0, p_1, \cdots, p_{n-1}\}$. Included in these n programs are test programs, applications programs, and testing programs. The test programs can be viewed as programs executed by a unit under test in response to the demand of the testing unit. Clearly, certain test programs can be expected to be more effective for different fault classes than for others. The applications programs, of course, perform the tasks or jobs for which the unit was defined. The testing programs are programs run by a unit when it takes the action of testing another unit. An Outcome Matrix which contained such a classification of programs or tasks and faults would greatly increase the potential applicability of a diagnosis theory to current fault tolerant computing designs.

In other words, given the units of a system, by so classifying all the tasks, and by specifying classes of faults that can inflict the units, a new, more applicable outcome specification can be described. It can be represented as a four dimensional array, and a planar description of it is shown in Figure 3. We will refer to this new outcome specification as the extended Outcome Matrix. Each unit now has a fault axis and a program axis, resulting in a u_i-plane and a u_j-plane; and the entries in the table are no longer binary values, but probability values from 0 to 1.

We are currently developing a general theory of diagnosability to exploit the information contained in the Extended Outcome Matrix. Our goal is to develop an enhanced diagnosability theory that is applicable to fault tolerant computing systems as complex as SIFT and FTMP. In general, a fault tolerant computing system continuously makes decisions about its own integrity. Such a decision-making process can be extremely complex, but the result of the decision-making process is relatively simple: a unit is either further utilized in computation or it is removed from the system. However, since removal of a unit effectively reduces the future reconfiguration capability and the potential performance, it must be done with considerable caution. In particular, for I/T faults, major complications exist beyond that which exist for permanent faults. A brief discussion of the nature of these complications would be appropriate here.

u_i \ u_j	f_0,p_0	f_0,p_1	f_0,p_2	...	f_1,p_0	f_1,p_1	f_1,p_2	...	f_{m-1},p_{n-1}
f_0,p_0	$a_{0,0}^{0,0}$	$a_{0,0}^{0,1}$	$a_{0,0}^{0,2}$	...	$a_{0,0}^{1,0}$	$a_{0,0}^{1,1}$	$a_{0,0}^{1,2}$	...	$a_{0,0}^{m-1,n-1}$
f_0,p_1	.								.
.	.								.
.	.								.
1									
f_{m-1},p_{n-1}	$a_{m-1,n-1}^{0,0}$		-		-				$a_{m-1,n-1}^{m-1,n-1}$

$a_{i,j}^{k,l}$ is the probability that unit u_i executing testing program p_j while inflicted by fault f_i assesses that u_j is faulty on the basis of its execution of program p_l while inflicted by fault f_k.

Extended Outcome Matrix

Figure 3

To begin, both intermittent and transient faults are usually lumped together. The reason for this is that they can exhibit similar error characteristics. But there are important differences from a diagnosis viewpoint. An intermittent fault is considered to always be present, but its error pattern is only detectable when it is in its active mode. With this type of fault, if the error pattern is detected at any time in a unit, then the unit is declared faulty. However, during the times between which the intermittent fault is in an active mode, the unit operates correctly; regardless, the unit is still classified as faulty and is removed from the system. Indeed, an intermittent fault could be such that its error pattern is almost never seen. Then even though a unit had such an internal fault, it could still be utilized most of the time, and, perhaps, should not be switched out. However, heretofore, there has been no means of analyzing such possibilities. This is just another ramification of treating an intermittent fault as though it were a permanent fault, because, clearly, for intermittent faults, unlike permanent faults, the overall degradation of the system resulting from the unit's removal might dominate the degradation resulting from intermittently faulty operation of the unit if it remained in the system. The decision-making process should consider this possibility for intermittent faults. The above can be more strongly stated for transient faults since transient faults are considered to be those induced from the external environment. The affected unit is not considered to be faulty. It could make an error because of the fault and it could perform flawlessly thereafter. Clearly, in this case the proper decision would not be to disconnect it from the system. Or, a transient fault could be so prevalent that the unit is almost always in error, and it should be switched out. Hence, it should be clear that a diagnosis strategy for I/T faults must have greater discrimination capability than that required for permanent faults.

It is our intention to use the Extended Outcome Matrix to evaluate diagnosis strategies of fault tolerant computing systems in terms of the overall effect on the system of a decision to remove/retain a unit when an error pattern caused by an I/T fault has been detected. In such systems, there must always exist some threshold for error levels which when exceeded results in the removal of a unit. This threshold must be a compromise which takes into account all possible uses and time criticalities of the system, and is probably at best near-optimal for any situation. In effect, such diagnosis schemes either implicitly or explicitly

make use of a dynamic discriminator function. The current state of the system, complexity of the program under execution, and penalty for an error must set the discriminant. When a unit exceeds an error level that has been set for it, it is switched out. Ideally, the more idle the system and the fewer the switched out units, the lower the discriminant should be. Alternatively, a damaged system, running a time-critical program, must have a high discriminant. Since the system cannot afford to lose any computing power, it can only switch out another unit when its error level becomes intolerably high. Hence, an understanding of the nature of the discrimination being employed by a system is crucial to analyzing its diagnosing capabilities. With a complete description of the discrimination and the Extended Outcome Matrix, we can then classify the performance of testing strategies of various fault tolerant computing systems. Accordingly, we are studying the concept of discrimination so that we can incorporate it into our model. However, the applicability of this, and in fact, all of our work depends entirely upon the availability of the Extended Outcome Matrix. Therefore, a methodology for generating the extended outcome matrix and a case study of an analysis using Intel 8080-based systems are fundamental parts of our program. We discuss our efforts along these lines in the next section.

6. Verification of a Methodology to Generate the Extended Outcome Matrix

By generating the Extended Outcome Matrix, we mean that we must obtain the $a_{i,j}^{k,l}$ values for the units in the system. Because of the complexity of the units and task programs, it is currently not realistic to attempt to predict the response of testing programs to units which can be inflicted by a multitude of different faults while executing a wide variety of programs. Hence, a laboratory system must be developed which can produce the probability values required for different interacting units in this system. This system must provide for experiments where one unit tests another unit while executing different programs in each and while subjecting both units to realistic fault situations. We are evolving a methodology which would be the basis of such experiments.

There are two issues which must be understood regarding our work in this part of the program:

First of all, if our methodology simply described a "brute-force" or "exhaustive" approach, we would really be accomplishing very little of general significance. However, we are not doing this. Instead, we are making some far-reaching conjectures about error pattern generation and emulation (we will discuss these further in the next section), and we are attempting to verify their validity with a case study and describe a feasible means of generating the Extended Outcome Matrix which exploits them.

Secondly, the laboratory system we have developed (see photographs in Appendix) is a complex network of computing devices and instrumentation which we have totally designed, built, and debugged. The instrumentation aspects of our system are intricate, finely timed combinations of hardware and software which must perform critical recording functions yet not interfere with the operation of the fault-free and faulty 8080-based units. It would be difficult to estimate the time we have spent on this system totally involved in mundane yet necessary tasks of hardware/software design and debugging but, easily, many hundreds of man-hours have been used in this work. Admittedly, it is a bare-bones design and there are many ways which we could enhance our system. Nevertheless, we know of no other system quite

like ours. Moreover, with it, we have already been able to conduct preliminary experiments which have allowed us to make some very interesting (and reaffirming) observations regarding error patterns (which will be discussed in the next section).

The key to our methodology is that we are employing a digital error representation of faults on bus lines. However, as we cannot "prove" the validity of our methodology, we must experimentally verify its validity. We are doing this by considering a case study wherein a unit is an Intel 8080 microcomputer module. A schematic of this module is shown in Figures 4 and 5. Such a case study gives us a controlled framework for evolving the methodology in that it provides highly important feedback to the evolution.

Our goal then is to generate an Extended Outcome Matrix where the Intel 8080-based system corresponds to both u_i and u_j. We intend to do so using a digital error representation of I/T faults. The processor communications bus has been selected as the system component upon which to develop the digital representation of the faults. To examine these bus lines, a gold processor is used for comparison. The gold processor is identical to the faulty processor except that the former is driven with a good power supply and the latter is driven with an I/T faulty power supply. We have only chosen to use I/T faults on the power supply because we currently lack other substantiated databases of I/T faults. Clearly I/T faults on the power supply line are realistic possibilities, and therefore, our work is not negatively affected by their use in the experiments. Moreover, as more fault data is produced (for example, from the current NASA funded Univac I/T fault gathering project), we can readily (and will gladly) utilize it in our study. As shown in Figure 6, a bus fault detector is used to determine differences between the buses. The bus fault detector has an output for each bus line, and indicates that the corresponding bus lines agree or disagree at any given time. To view the action of the bus during and after a fault arrival, an error catcher receives the bus difference signals and stores them for later retrieval by a monitor processor. A storage oscilloscope permits close examination of the error patterns. These digital error patterns are the "signatures" of the fault classes and programs on the specific unit being considered. In other words, if we change the structure of the unit (for example, suppose we were working with an AMD 2900-based system), for the same classes of faults and programs, the signature

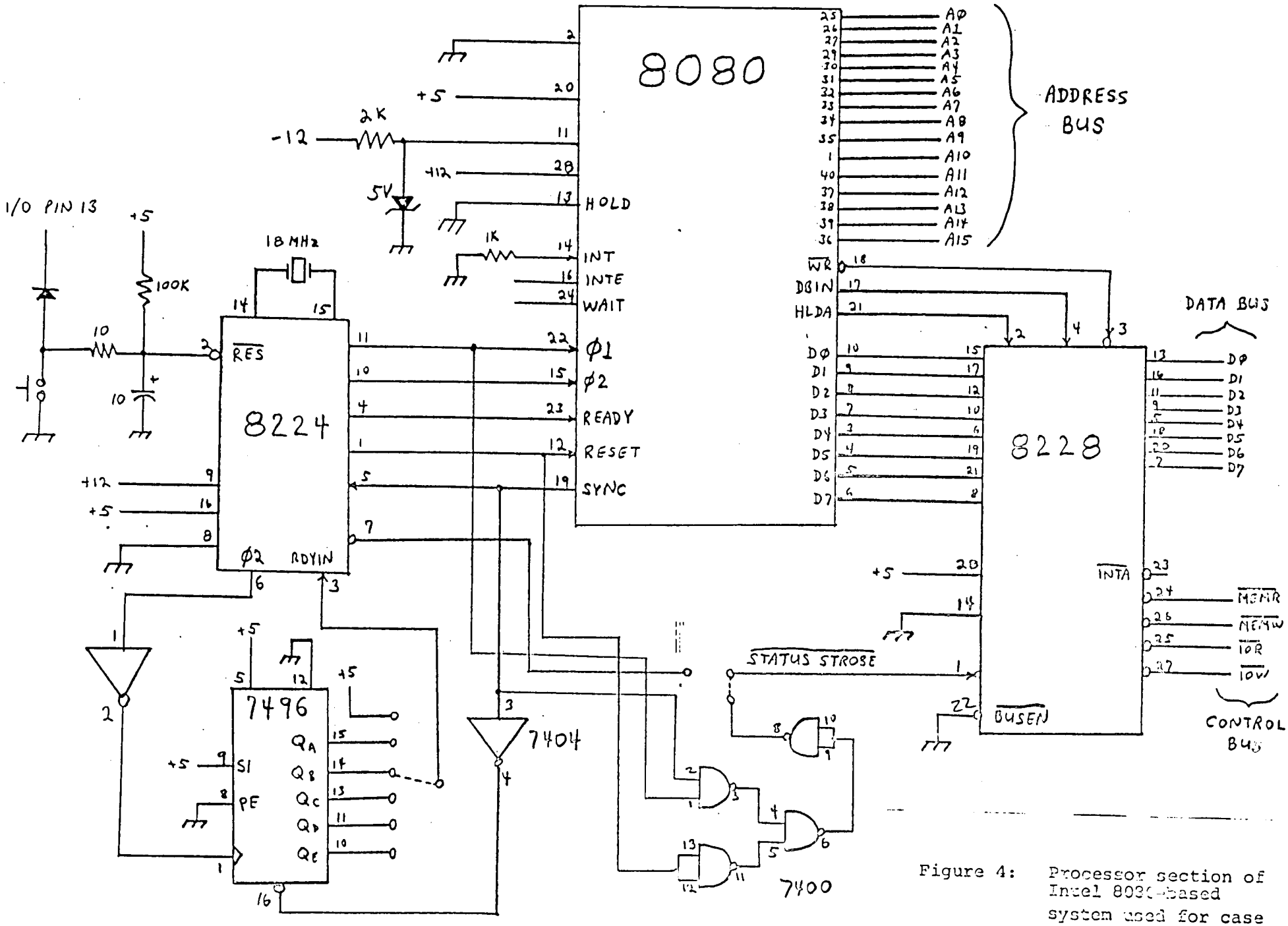


Figure 4: Processor section of Intel 8080-based system used for case study.

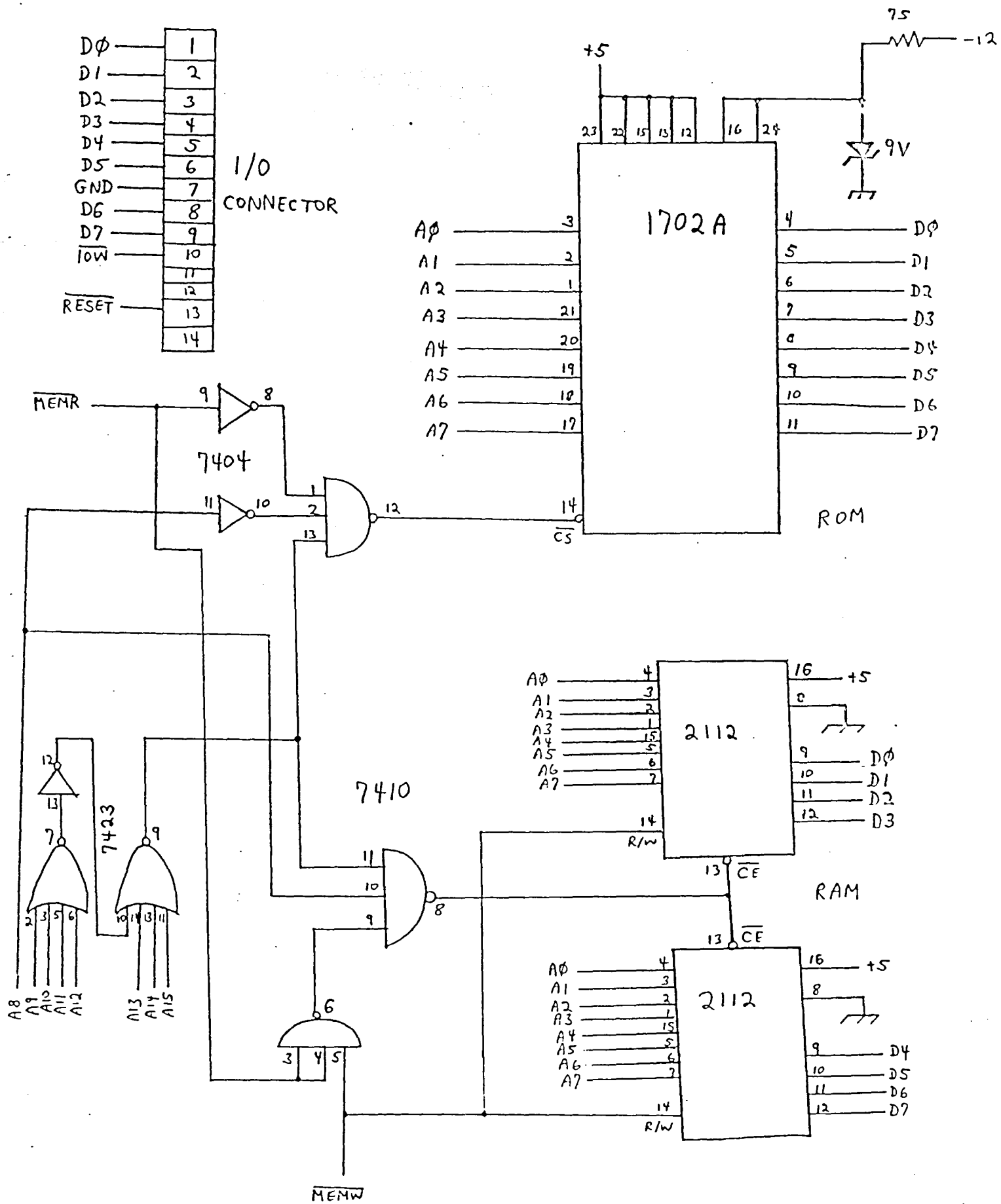


Figure 5: Memory section

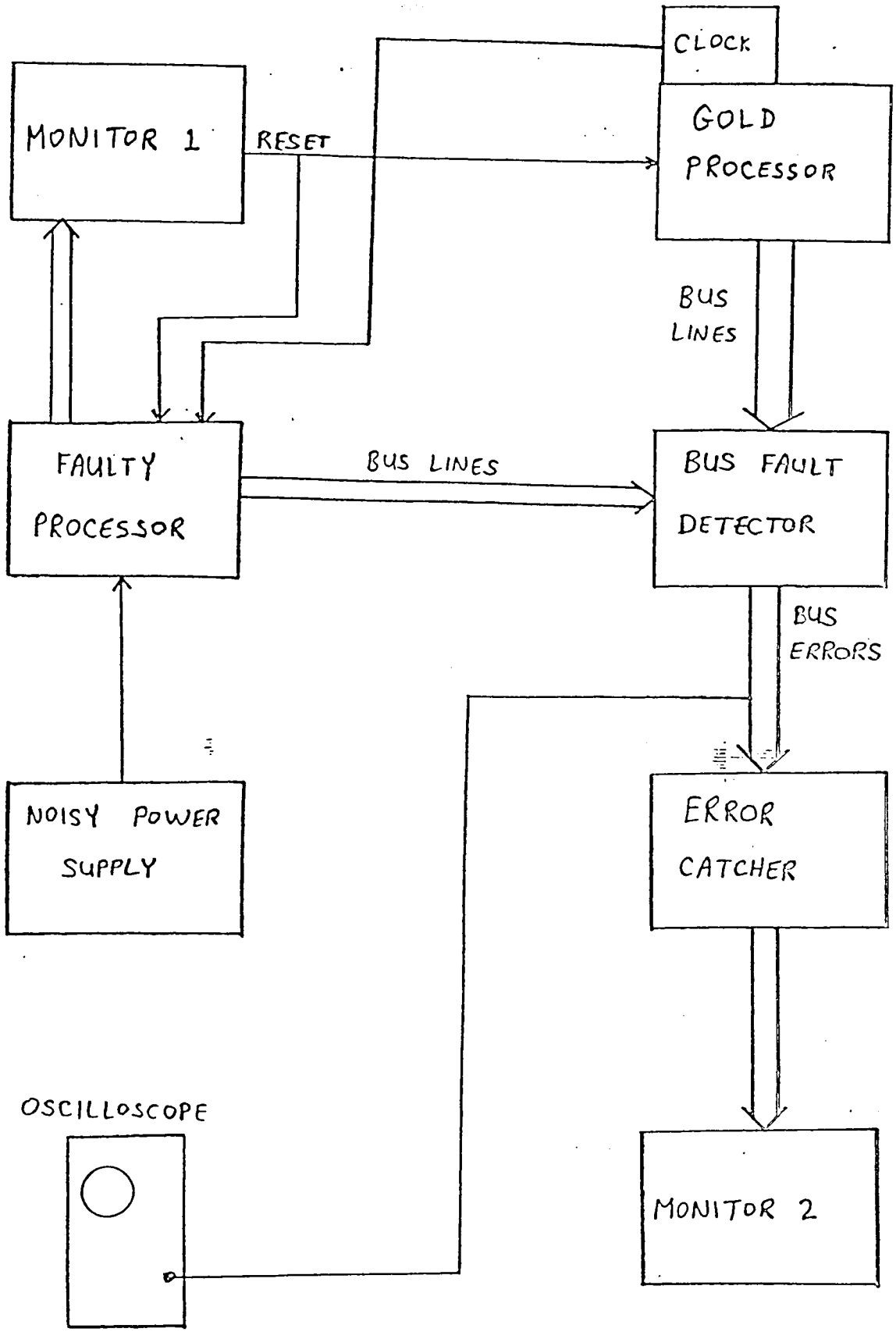


Figure 6: I/T Analog Fault to Digital Error Representation Experimental Set-up.

in terms of the error patterns would be expected to change. However, the methodology would remain the same; that is, we would still generate the signature for the classes of faults and programs of interest.

These error patterns are our digital representation of I/T analog faults. They are program dependent and unit dependent. For a specified unit, u_i, testing another specified unit, u_j, if we can collect their error patterns, we can then inject these error patterns in controlled situations into u_i and u_j to generate the Extended Outcome Matrix. Figure 7 shows such a set-up. The monitor processors preload the fault stores with error information modeling f_i and f_k. When the proper address is detected, the fault stores dump the information into the fault injection networks, duplicating the desired faults. U_i tests u_j, and informs its monitor processor of the test outcome. The set of such decisions is the basis of the Extended Outcome Matrix.

With the Extended Outcome Matrices for all interacting units, we can then apply the diagnosability theory we are now developing and measure the capability of the fault tolerant computing system to identify combinations of I/T faulty units, or suggest testing strategy modifications to enhance this capability.

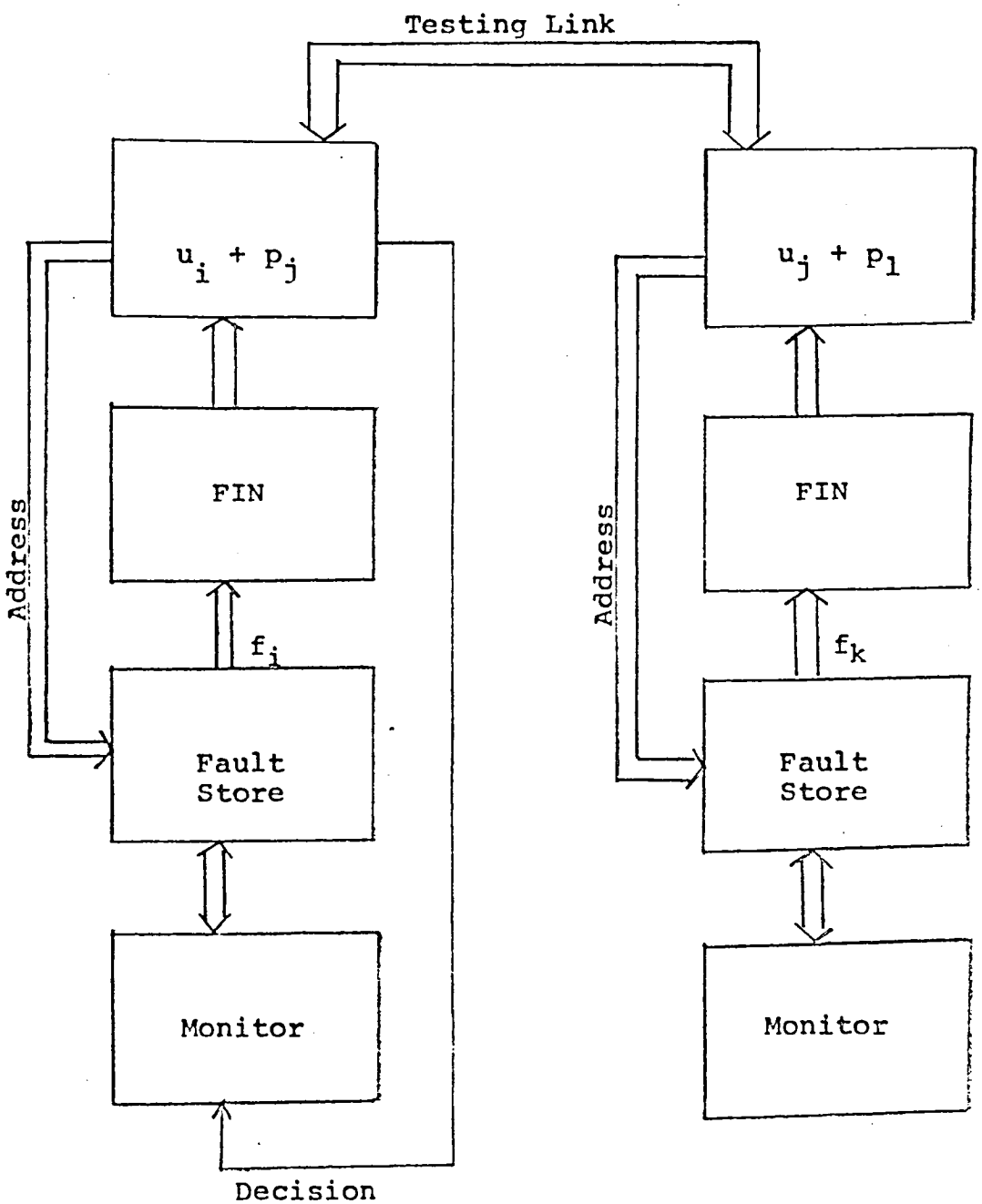


Figure 7: Extended Outcome Matrix Generation Experimental Set-up.

7. Preliminary Observations on Error Patterns

Data is now being collected from the experimental system schematically shown in Figure 6. (See Appendix for photographs.) The I/T fault being presently injected is power supply noise and a number of different programs have been used in the 8080-based gold and faulty units. We will describe in this section some preliminary observations corresponding to error patterns for one particular program, called INR1.0, which is a rather simple program being only three lines and seven bytes long. A listing of INR1.0 is given in Figure 11. The simplicity of INR1.0 is not important here, for observations similar to those we will now describe have also been made for much more complex programs; we use INR1.0 here only so that a relatively complete discussion of our observation can be made without a great deal of descriptive overhead.

To begin, when faults are injected and error patterns observed, there is the possibility regarding the outcome that, since we have no real control over the injected I/T fault (it is simply noise on the power supply), every fault pattern we observe for the same program will be different. If this were the case, then generating the entries in the Extended Outcome Matrix would be almost impossible, since to do so would mean that we would have to emulate a very large set of error patterns. However, this is not the case for simple combinational and sequential circuits. For such circuits, there are fault/error equivalences which allow all faulty modes of a circuit to be described with a reasonable number of fault equivalence classes. The question, then, is whether or not similar equivalences hold for error patterns on much more complex systems? We conjecture that, indeed, they do; and we offer the following observation relative to experiments with our system for INR1.0.

> Observation 1: For INR1.0, 250 runs were made to generate error patterns. For these runs, 46 distinct error patterns were observed. One specific error pattern accounted for 39% of that total. A second pattern accounted for 19%. The fact that two patterns account for over half (58%) of all the patterns generated indicates that some type of error pattern equivalence notion is relevant and that successful error data compression is feasible. The situation is actually better than the above figures show. A number of different

error patterns are of two families. Each family differs in only a few places. The differences within a family are being explored, and it is believed that to a large extent, each family will produce the same results on the test system. This implies that then a few families can potentially be used to account for almost all of the error patterns generated.

A family of error patterns is shown in Figure 8. The possible types of errors are address, data, and control errors. The values range from 0 to 3, giving the number of error types in each window. Each pattern in this family lasts for 47 cycles. It can be seen that pattern #59 covers all others in the family, in the sense that no other pattern has more errors than pattern #59 at any given time, and each member of the family is basically pattern #59 with a few missing errors. In general, each family can be covered by one of its members. The fault/program interaction can be modeled with a set of families, and each family of the set can be represented by one pattern. Hence, the overall interaction information can be reduced sufficiently to be usable while still retaining sufficient information to permit distinguishability.

The next issue which must be addressed is, to generate the entries of the Extended Outcome Matrix, we must emulate the error patterns. What is the complexity of doing so? Again, for combinational and sequential circuits, it is well-known, for example, that simple prime stuck-type faults exist and can be injected to cause the circuits to display its faulty functional modes of operation. Can something similar be done for error patterns on much more complex systems? We conjecture that, indeed, it can; and we offer the following observation relative to experiments with our system for INR1.0.

Observation 2: Detailed examination of the most common pattern described in Observation 1 shows that a single error on the bus generated the entire error pattern. Figure 9 shows the complete bus pattern for type #59 (302). The first frame shows a single error on D6. This one error is sufficient to generate the entire pattern. The two processors are thrown out of synchronization, and they converge 47 cycles later. From this it is evident that

the error patterns consist of two portions. The beginning part is the prime excitation period (PEP), and the pattern subsequent to that is a result of that excitation. During the PEP, the fault is either directly acting on the bus lines or for the very first time the results of the fault are propagated to the bus. After the PEP, the faulty processor is performing correctly, but is not in agreement with the gold unit because the faulty unit has diverged to a different path of execution. In the specific case of Figure 9, the paths do converge later, but this will not always happen with larger programs. (This has been experimentally verified with a program called FMP2.0). Therefore description of the PEP and the arrival location is sufficient to describe the entire error pattern. In the case of pattern #59, the PEP is just one instruction in length, and this pattern is the most prevalent (39%) of all the patterns.

The above leads to a final conjecture which we are now examining: for specified faults and programs, a high percentage of all the error patterns produced can be modeled by a small number of single instruction errors. If proven true, this would indeed be a remarkable result. Our sought after compression would then come naturally from error pattern families which could be reproduced by single byte errors. This would have to be related in some way to program sensitivities as implemented on the target processor. However, our preliminary investigation at this point gives no clues as to how it could be predicted that, for example, 39% of all patterns could absolutely be duplicated by a single bus error at a single address.

We can then summarize our I/T fault modeling hierarchy as shown in Figure 10. The true model of the I/T fault is an analog model. The most basic digital representation of this I/T analog model is a complete detailing of fault-free, stuck-at-one, or stuck-at-zero conditions on every line in the system. (Such a model would be used with a gate level simulator.) A more tractable, compressed form of this digital representation would be the error pattern on the bus lines. This, however, leads to an even more compressed model represented by a PEP, which seems to correspond to a single (or, perhaps, a few) instruction failures. The implication of this hiearchy is that the fault store for emulation of error patterns can be quite simple.

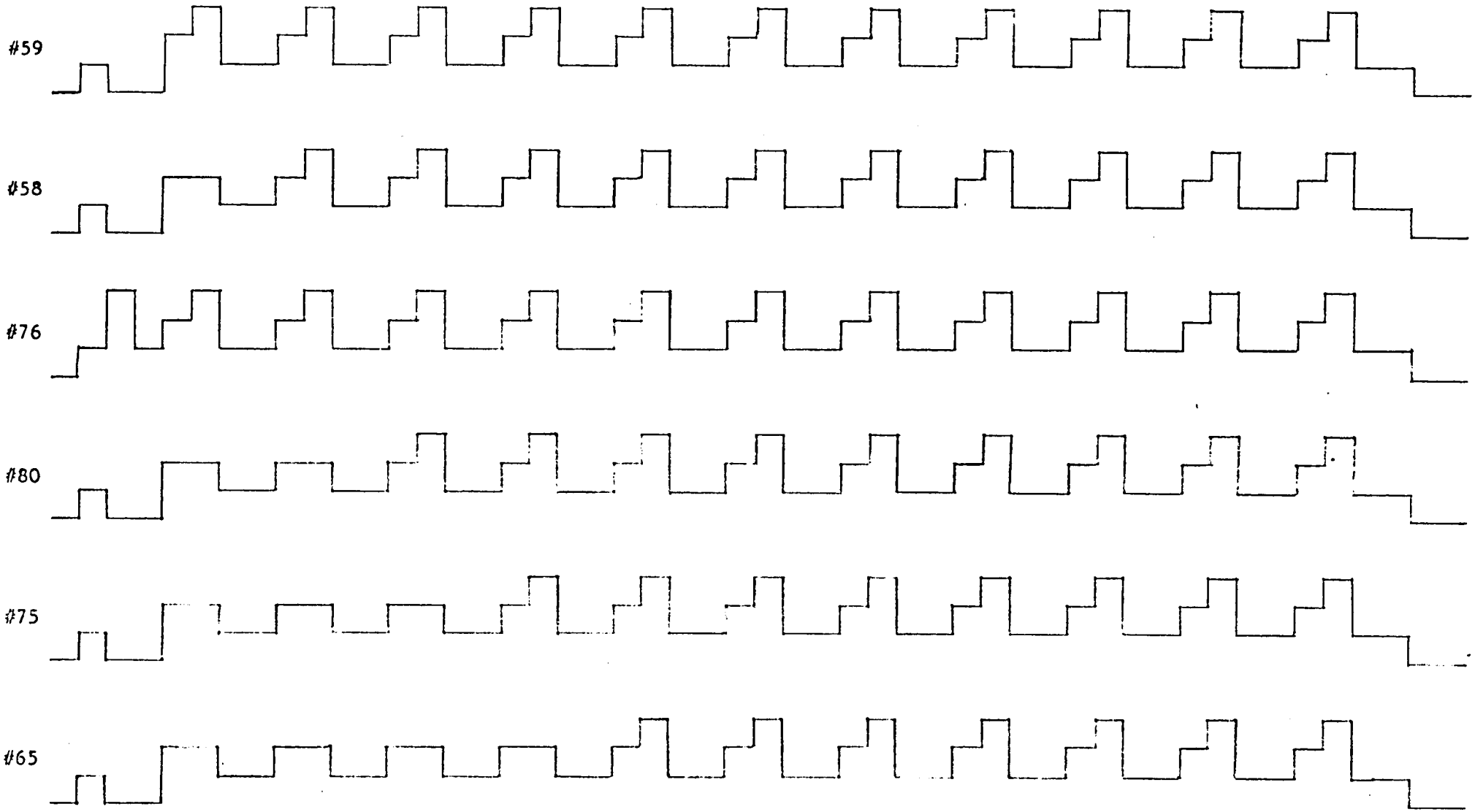
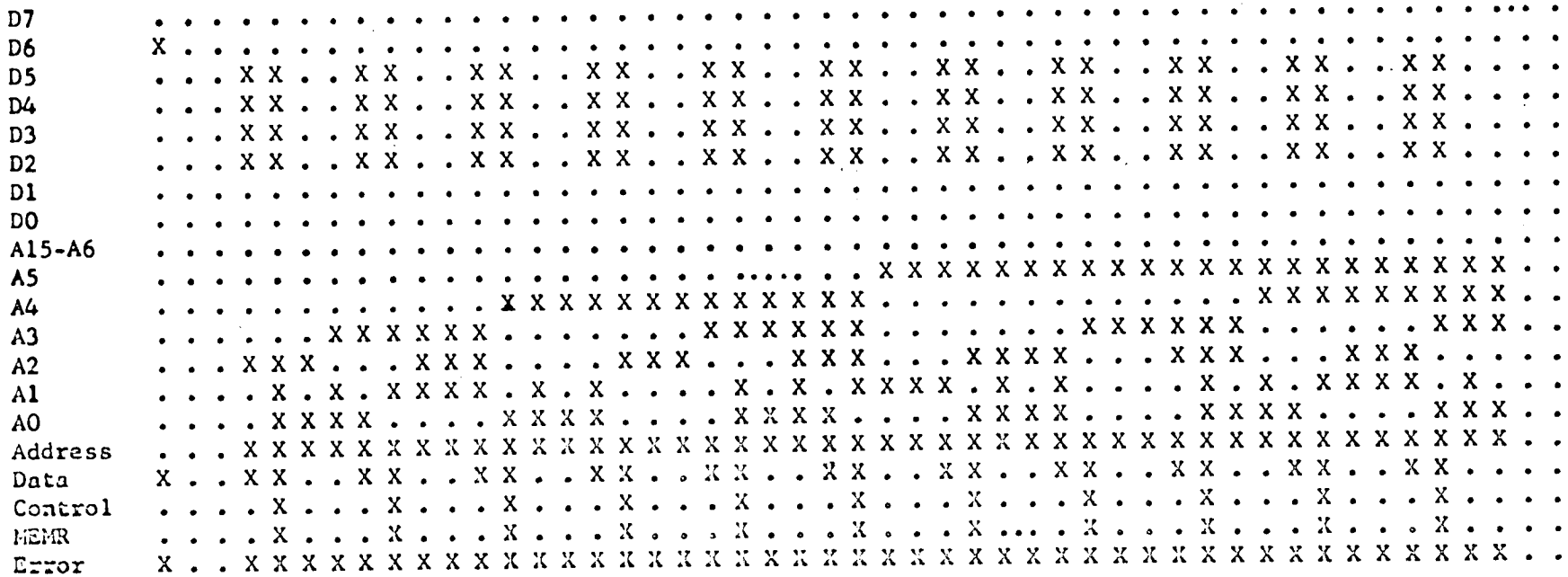


Figure 8: A family of error patterns for program INR1.0



Composition of patterns #302, 286, 292, 297, 302, 310

X = bus difference

Figure 9: Complete bus error pattern for type #59 (302) for program INR1.0

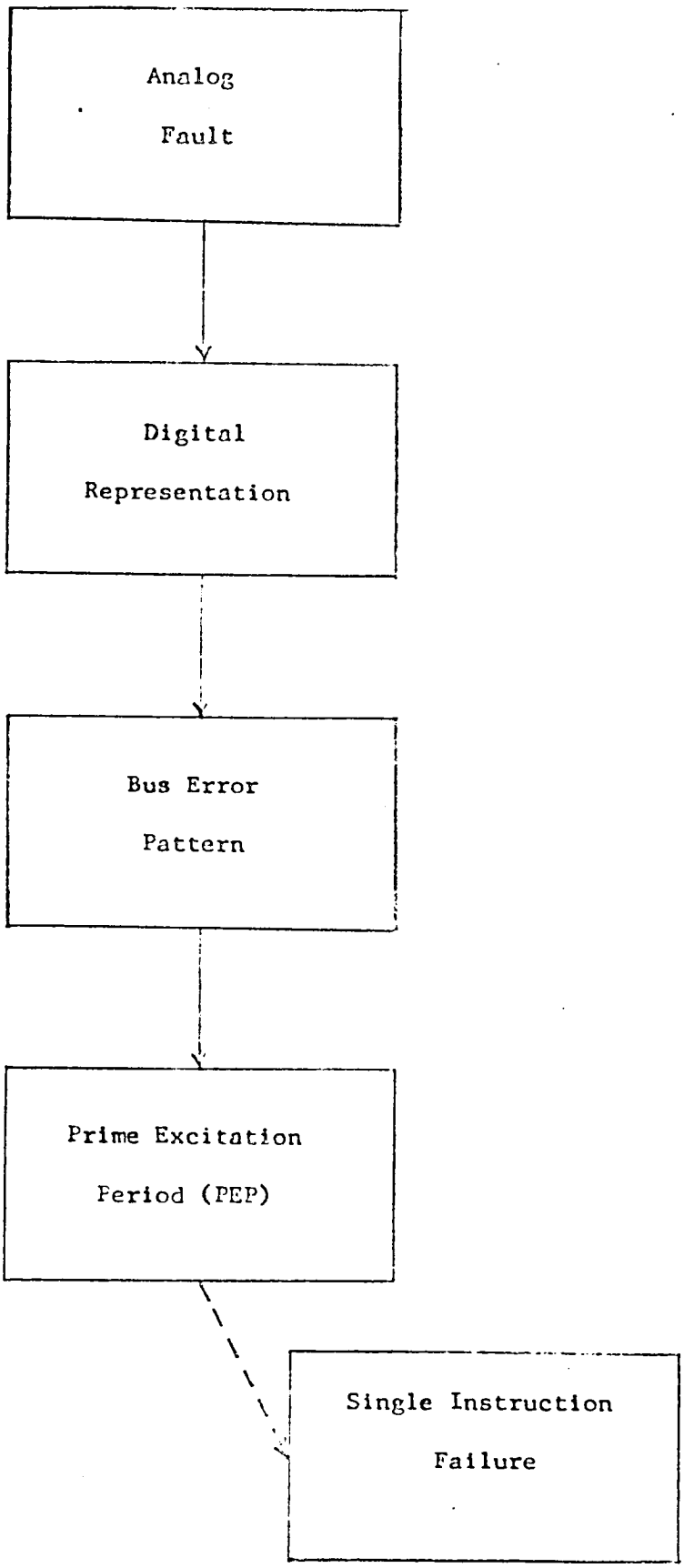


Figure 10: Fault Modelling Hierarchy

```
0001 0000                       ;SIMPLE PROGRAM FOR USE WITH
0002 0000                       ;
0003 0000                       ;ERROR PATTERN EXPERIMENTS
0004 0000                       ;
0005 0000                       ;
0006 0000                       ;INR1.0
0007 0000                       ;
0008 0000                       ;JULY, 1979
0009 0000                       ;
0010 0000                       ;
0011 0000                       ;ROBERT GLASER
0012 0000                       ;
0013 0000                       ;
0014 0000                       ;
0015 0000                               ORG     0
0016 0000 3C                    START:  INR     A
0017 0001 C3 00 00                      JMP     START
0018 0004                       ;
0019 0004                       ;
0020 0004                       ;
0021 0004                       ;       INTERRUPT LOCATION
0022 0004                       ;
0023 0004                               ORG     38H
0024 0038 C3 00 00                      JMP     START
0025 003B                       ;
0026 003B                       ;
0027 003B                       ;
0028 003B                       ;ALL UNSPECIFIED ADDRESSES ARE 00
```

Figure 11: INR1.0 program

8. Conclusion

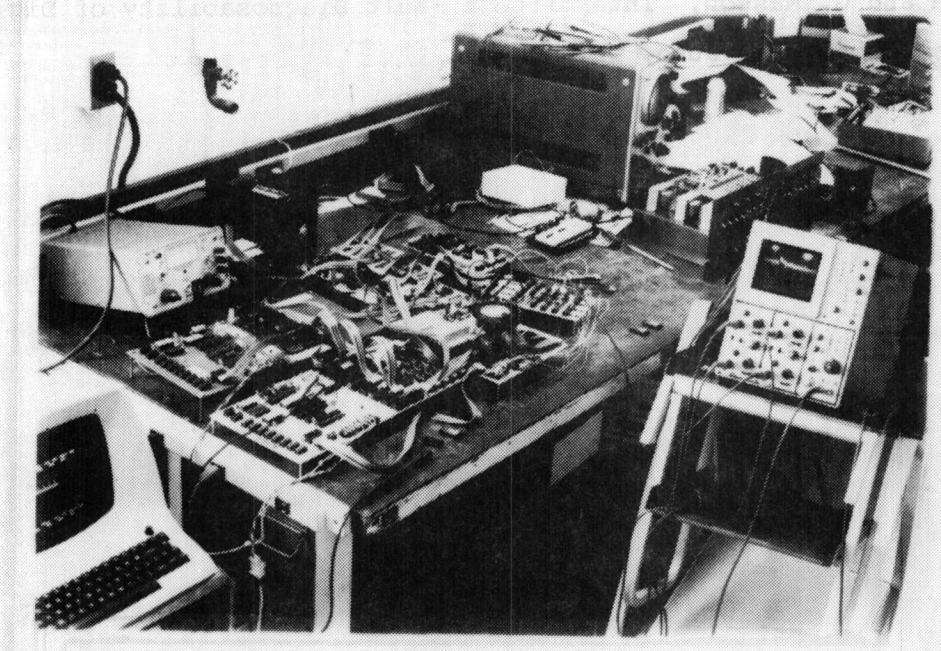
We believe that the multi-faceted program being conducted in the Electrical Engineering Department at Johns Hopkins is in agreement with NASA's goals in the fault tolerant computing area. The parallel, tightly coupled theoretical and experimental efforts in which we are engaged can truly make a contribution to the validation process of fault tolerant computing systems. The eventual diagnosability analysis of systems of the complexity of SIFT and FTMP is achievable, and on the basis of such analysis the reconfiguration capabilities of these systems can be emulated. Such evaluations will make fundamental contribution to the validation process of fault tolerant computing systems.

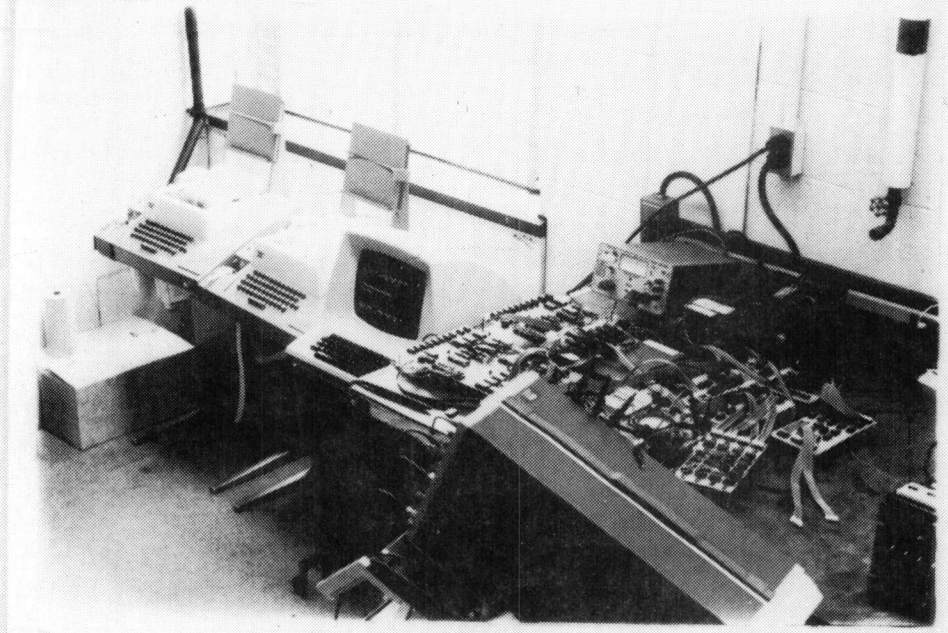
References

1. "Validation Methods for Fault-Tolerant Avionics and Control Systems - Working Group Meeting I, NASA CP-2114, 1979.

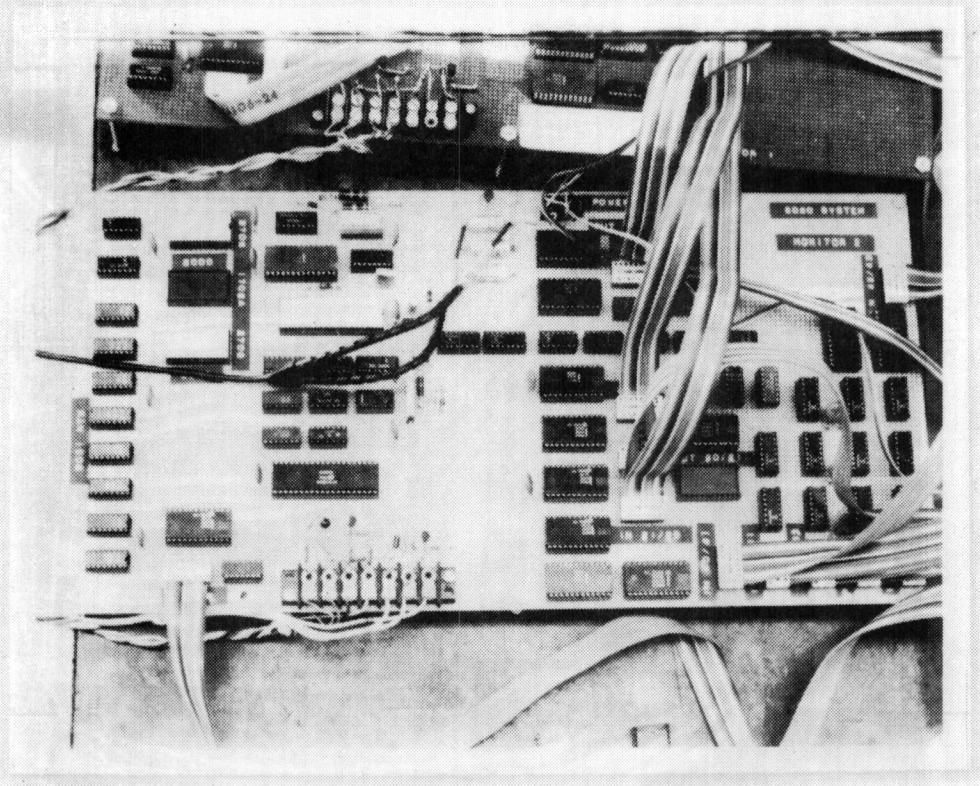
2. S. Mallela and G. Masson, "Intermittent Fault Diagnosability of Digital Systems," IEEE Trans. Computers, June, 1978.

Appendix

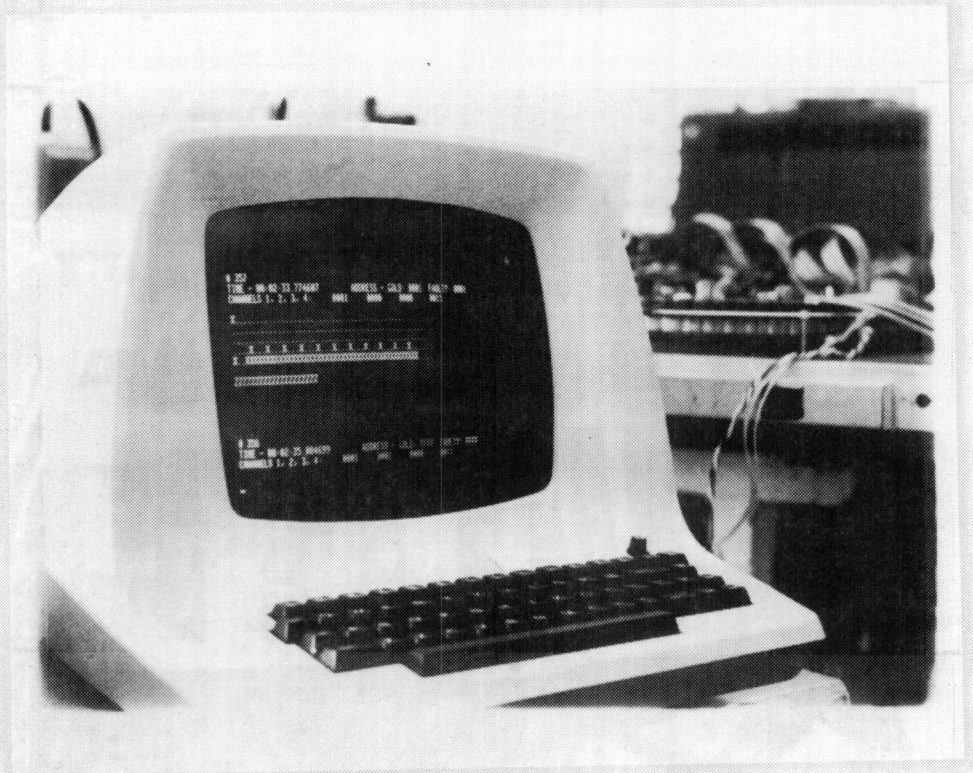




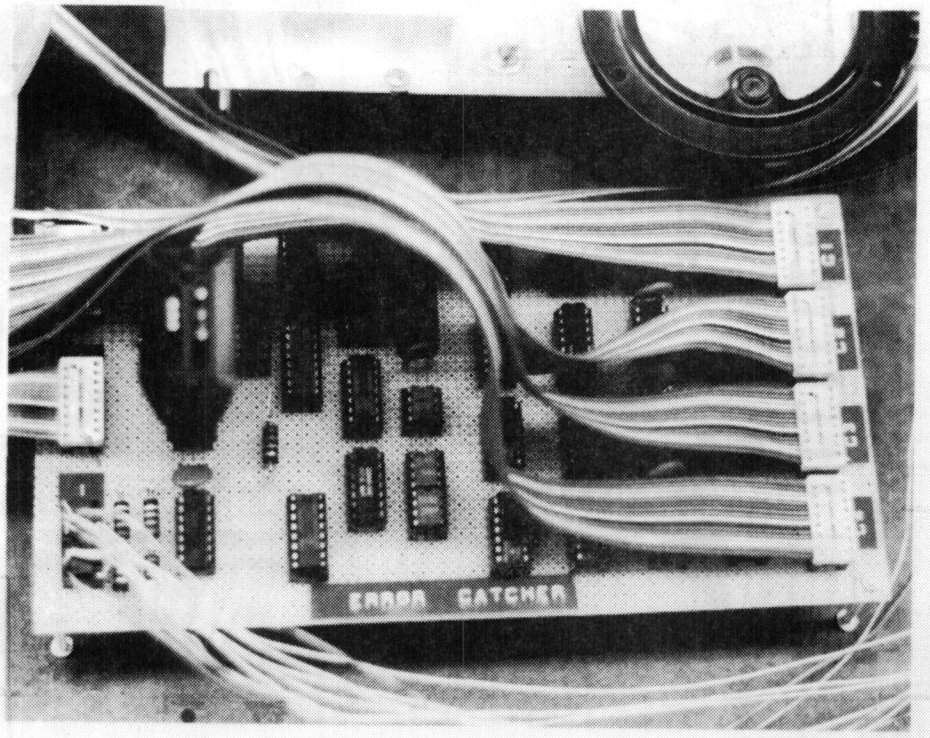
The JHU Experimental System



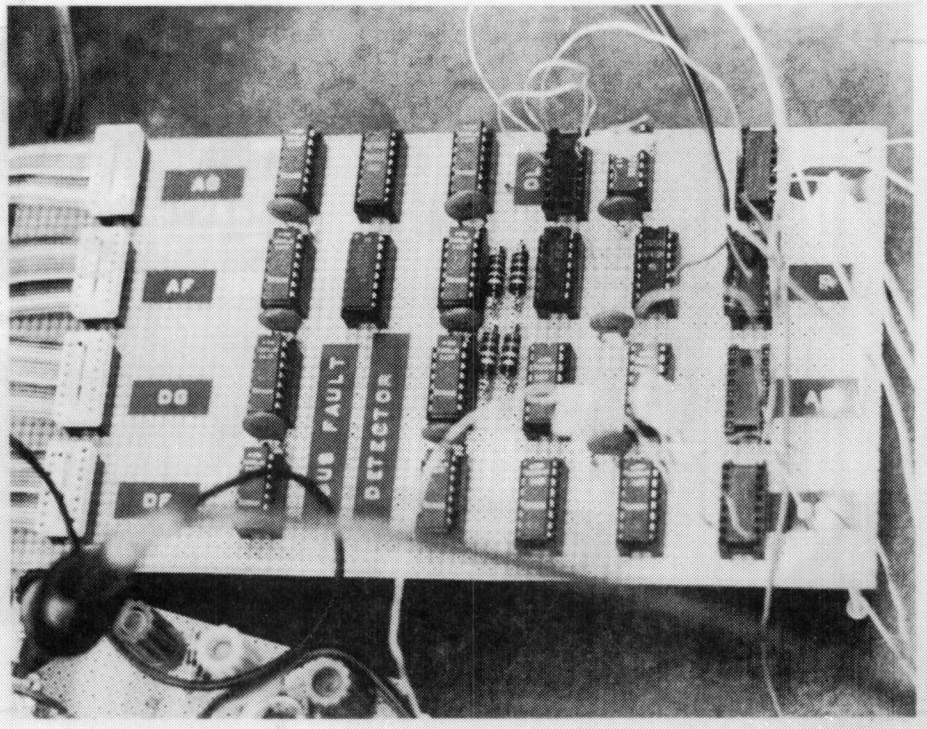
A Monitor



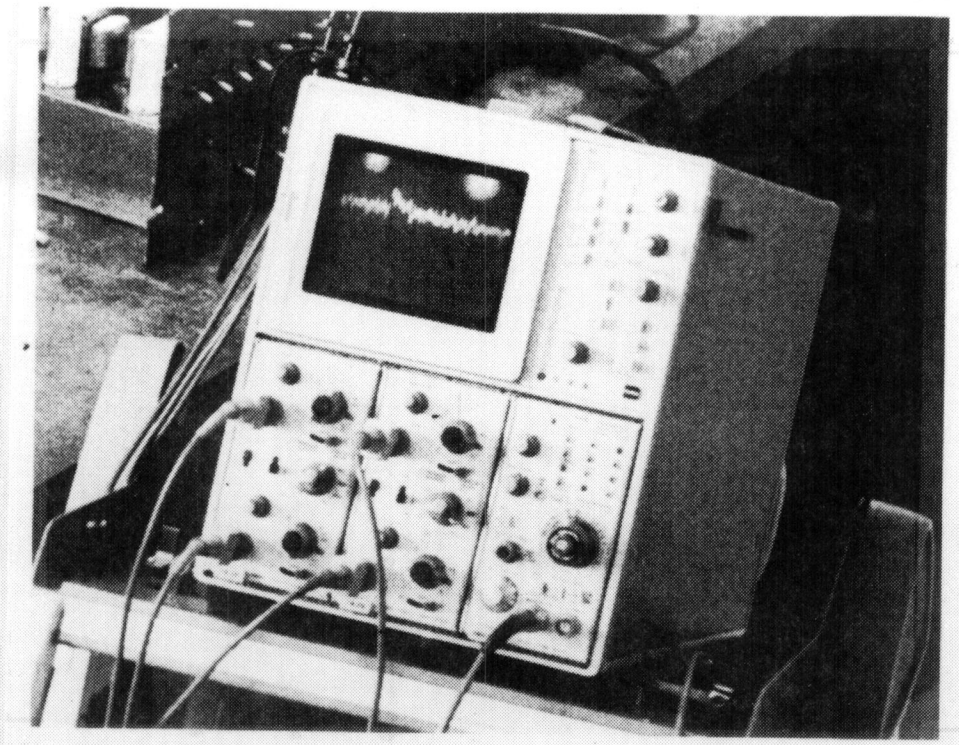
Scope-Terminal Displaying Error Pattern



The Error Catcher



The Bus Fault Detector



Storage Oscilloscope Displaying I/T Analog Fault

1. Report No.	2. Government Accession No	3. Recipient's Catalog No.
NASA CR-159229		

4. Title and Subtitle	5. Report Date
INTERMITTENT/TRANSIENT FAULTS IN COMPUTER SYSTEMS-- EXECUTIVE SUMMARY	April 1980
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Gerald M. Masson	
	10. Work Unit No.
9. Performing Organization Name and Address The Johns Hopkins University Electrical Engineering Department Baltimore, MD 21218	11. Contract or Grant No. NSG-1442
	13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

NASA Technical Engineer: S. J. Bavuso
Executive Summary

16. Abstract

An overview of an approach for diagnosing intermittent/transient (I/T) faults is presented. The goal of this study is to develop an interrelated theory and experimental methodology which can be used in a laboratory situation to measure the capability of a fault tolerant computing system to diagnose I/T faults.

To the extent that such diagnosing capability is important to reliability in fault tolerant computing systems, this theory and supporting methodology will serve as a foundation for validation efforts.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Intermittent, Transient, Fault Modeling, Error Models, Fault-Tolerance, Computer Systems	Unclassified - Unlimited Subject Category 66

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	36	

* For sale by the National Technical Information Service, Springfield, Virginia 22161

DO NOT REMOVE SLIP FROM MATERIAL

Delete your name from this slip when returning material to the library.

NAME	DATE	MS
[redacted]		

NASA Langley (Rev. Dec. 1991) RIAD N-75